EXHIBIT 2

Virtualization Top-Level Design and Software Requirement Specifications

Project:

Requirements Manager: Sarel Altshuler

Document ID: _____ **Revision:** 1.0 **Date of issue:** 1/29/01

Reference: C:\WINDOWS\TEMPORARY INTERNET FILES\OLK51\VIRTTLD.DOC

Document status: _____

Page count: 39

Abstract:
This document presents the virtualization top-level design.

Approved:

Name	Date	Comments
Gadi Erlich **Roni Sayag**		

Distribution for Review:

Name	Phone	Review Date
Gadi Erlich Erez Waisbard Alexander Tsirilin Felstine Eyal Roni Sayag		

Table of Contents

Table of Figures

1. General

Figure 1 illustrates the top-level design of the virtualization module.

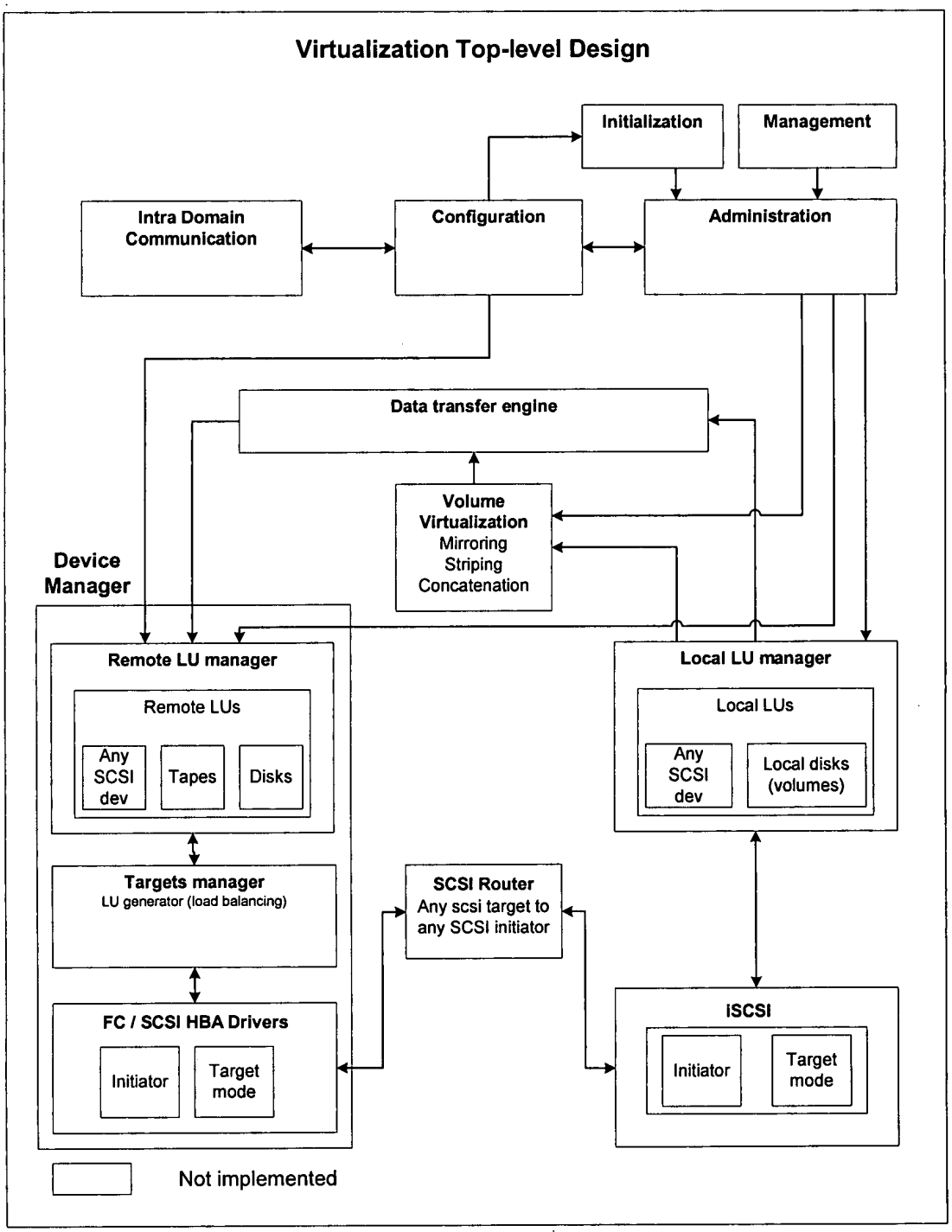

Figure 1 Top-level design

This section gives a brief presentation of each module in the system while next sections provide more details on each module.

1.1. Initialization

Boots the virtualization module using parameters taken from the configuration and sent to the administration.

1.2. Management

Provides a user interface for all configuration operations and status indications. The module sends the new configuration changes to be executed by the administration module.

1.3. Administration

Performs, monitors and synchronizes the actual execution of each configuration/operational changes in the system.

1.4. Configuration

Saves restores and synchronizes all configuration parameters (disks, volumes, access permissions).

1.5. Intra domain communication

The Intra domain communication module provides a generic interface for spreading information within the domain. It is mainly used to negotiate and spread configuration parameters and operational changes (fail-over) between all SDCs within the domain.

1.6. Data transfer engine

Performs the actual data transfer between iSCSI virtual targets, presented by the SDC, to the physical remote lus.

1.7. Volume virtualization

Translates requests directed to virtual volumes into a list of operations that should be performed on physical devices in order to fulfill the request. This module implements all virtualization techniques (striping, mirroring, concatenation).

1.8. Local lu manager

Implements and exposes local lus that represent virtual disks, tapes and any other device.

1.9. SCSI router

One-to-one translation between FC/SCSI attached devices to iSCSI devices. There is no virtualization in this process. Selected targets are presented transparently on the iSCSI side.

1.10. Device manager

The device manager is the collection of remote lu manager, target manager and HBA drivers modules.

1.11. Remote lu manager

Discovers controls and operates remote lus (disks, tapes and any other SCSI device).

1.12. Target manager

Collects and unite targets paths from different host bus adapter. Send these targets to the remote lu manager for remote lu discovery. Performs load balancing between different paths to the same target.

1.13. Host Bus Adapter (HBA) drivers

Low-level drivers that provide simple interface to send (SCSI initiator) and/or receive (SCSI target) SCSI commands.

2. Device manager

The device manager consists of all modules that handle the operation of SCSI devices connected to the SDC using parallel SCSI or Fibre channel. The sub modules in the device manager are the HBA drivers, target manager and remote lu manager. Subsequent sections will explain each module in more details. Figure 2 gives an example of device manager objects in a system with two host bus adapters, one disk and one RAID controller with two logical units. The example illustrates the objects in the system and the relationship between them.

A separate object represents each Host Bus Adapter. The HBA objects are generated and owned by the HBA driver module. A separate "target path" object represents each SCSI target seen by each HBA. In this example there are two HBAs, each one discovered two different "target paths" thus we have two "target paths" for each device. A "target" object represents each target in the system and a list of "target paths" is associated with each target. Target paths are generated by an HBA object and are exposed to the target manager. Targets generated and maintained in the target manager module.

For each target, lu discovery process is performed and a separate lu object is generated. In this example, on the disk target we found one "disk lu" and on the RAID target, we discovered two lus, lu0 and lu1. The discovery and maintenance of the lus is performed in the remote lus software module.

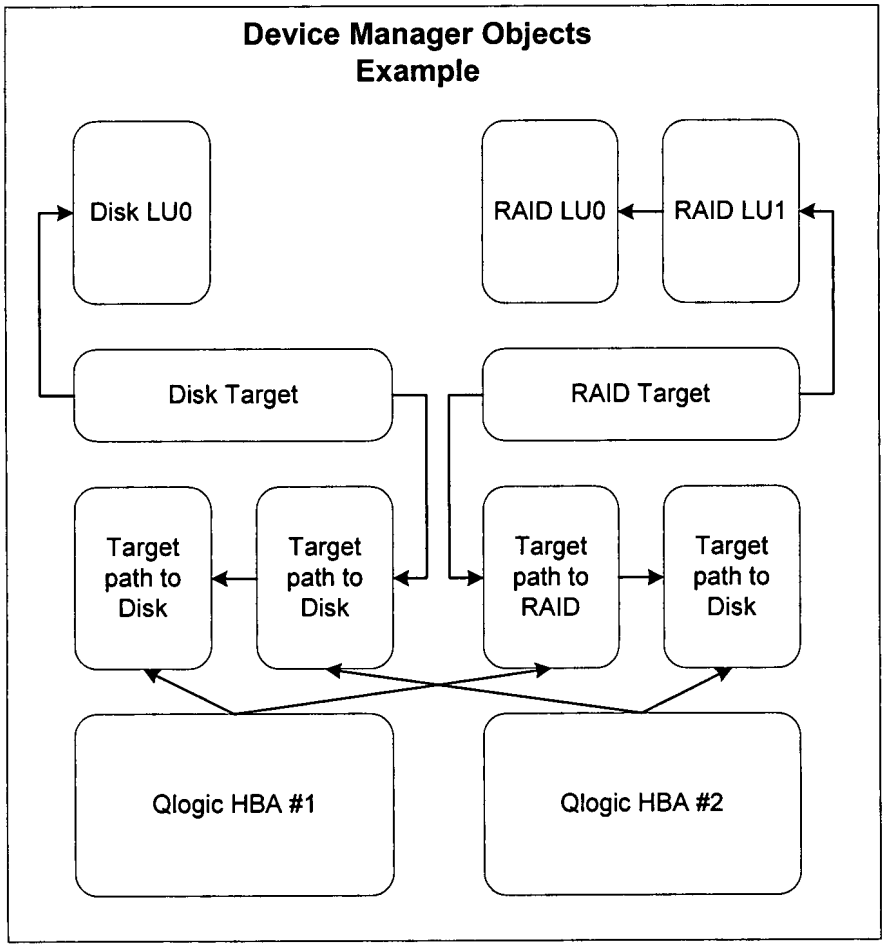

Figure 2 Device manager objects example

3. HBA Drivers

The HBA drivers implement a simple generic SCSI interface to the reset of the system. The interface allows sending and receiving SCSI commands with only basic error recovery services. The interface also provides basic discovery services and exposes a list of SCSI target paths to the upper layer. It is the responsibility of the upper layers to unite target paths seen by more than one interface to one logical target and to discover the different lus on this target.
At first stage, we will implement fibre channel and parallel SCSI initiators. For "Box A" we will add also iSCSI initiator in the iSCSI module.

3.1. Basic objects

3.1.1. Host Bus Adapter

An object that represents the physical interface to the storage network. Host bus adapters are discovered over the PCI buses at system initialization or even "on the fly" if we support PCI hot-plug. The physical interface might be either parallel SCSI, Fibre channel or iSCSI.
In a perfect world, there are several layers to the HBA object. The basic object represents a generic HBA. The next level discriminates between FC, SCSI and iSCSI HBAs and the last level discriminates between HBAs from different vendors as illustrated in Figure 3.

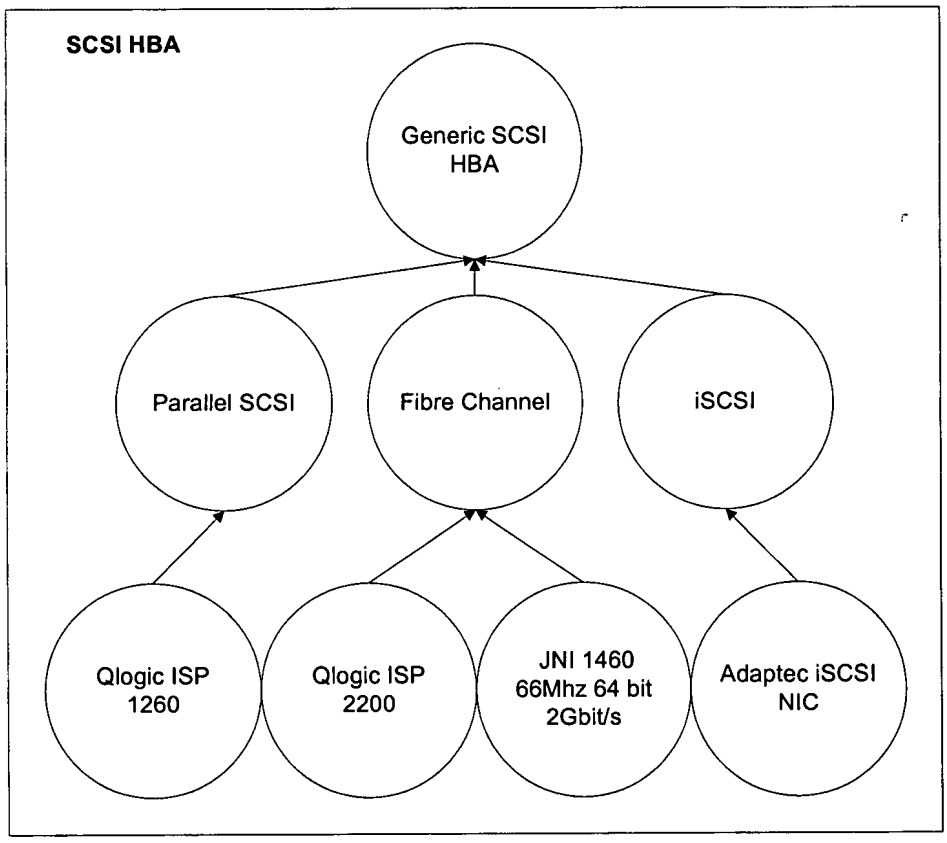

Figure 3 HBA hierarchy

3.1.1.1. HBA Interface

3.1.1.1.1. HBA constructor

Creates a new HBA and initializes the underlying hardware. At the end of this process, the HBA is ready to send / receive new commands to one of its target paths.

3.1.1.1.2. Discover target paths

Get a list of all targets seen by this interface. This interface should be bi-directional. The client of the HBA driver might ask for a list of all targets. On the other hand, the HBA should notify the target manager whenever there are changes in the target paths (targets added or removed on-line). The HBA notifies the target manager of new created target paths (see Target manager).

3.1.1.1.3. Get maximum transfer length

Returns the maximum length of data that can be sent using this HBA.

3.1.1.1.4. Get / Set self address

Set or get the SCSI address of the HBA.

3.1.1.2. Data members

3.1.1.2.5. Container for all target paths

A list of all currently discovered target paths.

3.1.1.2.6. Self address

SCSI address or the adapter.

3.1.1.2.7. Hardware control parameters

Variables used to control the HBA hardware. This might include pointers to H/W registers in the HBA and queues for requests and responses.

3.1.2. Target path

Target path object represents a SCSI target as defined in T10 SAM specification. In fibre channel, "target path" is uniquely identified by its **port name**. Target path is the final destination for a SCSI command.

3.1.2.1. Interface

3.1.2.1.8. Target path constructor

Creates new target path with port name as a parameter.

3.1.2.1.9. Send SCSI command

Allocate resources for a new SCSI command and send it's Command Descriptor Block (CDB) to the specified target. In certain situations, this interface will trigger the execution of the whole SCSI phases up to SCSI status phase.

3.1.2.1.10. Send SCSI task management command

Send SCSI task management command (abort task, abort task set, clear task set and target reset) to the specified target path.

3.1.2.1.11. Exchange data in/out

Send data to the target in write command and receive data from the target in read command.

3.1.2.1.12. Receive RTT

Receive an indication from the target that data can be sent. This indication is relevant only for write command using fibre channel interface.

3.1.2.1.13. Receive SCSI response

Receive transport response; SCSI response and optionally sense information in case of "check condition" SCSI status.

Remark: Not all SCSI chips allow splitting the execution of a SCSI command to its different phases (command, data in/out and status). In this case, the interface is simpler and there is one interface that executes a complete SCSI transaction. When the "driver user" initiates a SCSI command, all parameters should be supplied including the data for the command and pre-allocated space for the response.

3.1.2.2. Data members

3.1.2.2.14. Target path id

Unique id of the target path: port name in fibre channel.

3.1.2.2.15. Target role

SCSI target, initiator or both.

3.1.2.2.16. Active commands

List of currently active commands. The commands are actually "remote transactions" objects as described in 5.1.5.

3.1.2.2.17. Dormant commands

Commands waiting to be executed. The commands are actually "remote transactions" objects as described in 5.1.5.

4. Target manager

Target manager is the connecting link between the "remote lu managers" to the SCSI HBA drivers. All commands to the SCSI drivers pass through the target manager.
The target manager performs the load balancing and fail over between the different "target paths" to the target. The load-balancing algorithm is yet to be decided.
The target manager should also perform error recovery operations that are relevant to SCSI targets. An example for such operation is sending "target reset" task management command.

4.1. Basic objects

4.1.1. Target manager

The target manager is a single object that creates and holds all the targets in the system.

4.1.1.1. Interface

4.1.1.1.18. Accept target path

Accept a new created target path from one of the HBAs. The target manager should check if this target path is to an existing target or to a new target. In case of a new target, the target manager creates a new target and adds the target path to it. If the target is already created, we should only add the target path to it.

4.1.1.1.19. Remove target path

Remove to target path from one of the targets.

4.1.1.2. Data members

4.1.1.2.20. Targets container

Container that holds all targets in the system.

4.1.2. Target

Target object represents a SCSI physical target that may have more than one port (target paths). In fibre channel, target is uniquely identified by its **node name**. When a remote lu sends a SCSI command, it sends it to the target that contains this specific lu.

4.1.2.1. Target Interface

4.1.2.1.21. Target constructor

Creates new target. In fibre channel the supplied parameter is the node name.

4.1.2.1.22. Send command

Send SCSI command to the target using one of its target paths. The command is a "remote transaction" as described in 5.1.5.

4.1.2.1.23. Send task management

Send SCSI task management command to the target using one of its target paths. The command is a "remote transaction" as described in 5.1.5.

4.1.2.1.24. Add target path

Add another target path to the same target.

4.1.2.1.25. Remove target path

Remove existing access path from the target in a cases were the target is no longer accessible through it.

4.1.2.1.26. Change load balancing algorithm

Turn on/off or change load-balancing algorithm for the specified target.

4.1.2.2. Data members

4.1.2.2.27. List of target paths

List of all target paths from which we can access the specified target.

4.1.2.2.28. Target id (node name).

Target unique identification. In fibre channel, it is recommended to use node name as target identification.

4.1.2.2.29. Load-balancing parameters

Load-balance algorithm. History variables for calculating next "target path" to use.

5. Remote LU Manager

The remote lu manager discovers and controls the operation of all SCSI devices attached to the SDC. The module hides all SCSI knowledge from the upper layers and provides an intuitive interface for the specific device type. At the heart of the module, lies the "remote lu manager" object. This object discovers the lus over the targets and act as a container for all these lus. For each lu discovered, an lu object is created. Once the lu object is created, it is responsible to handle all request to this lu and guarantee the healthy operation of it including error recovery operations specific to the device type.

5.1. Basic objects

5.1.1. Remote lu manager

As described above, the "remote lu manager" discovers, holds and manages all remote lus.

5.1.1.1. Interface

5.1.1.1.30. Accept new target.

Called from the targets manager object to inform the remote lu manager of new discovered target. The remote lu manager should make an "lu discovery" on this target in order to create new lus. In my opinion, according to the current architecture, all discovered lus should be "created". A slightly different approach is to consult the configuration module whether the specific device is part of our configuration and thus should be "created". At this stage, I can't see situations in which we discover for example disk and don't use it. Even if the specific SDC does not use this disk, another SDC is probably using it and this SDC should be prepared for SDC take over in case of a failure. Even if none of the SDCs in the domain are using the disk, it should be presented to the user with all it's capabilities.

5.1.1.1.31. Get remote lu list

Upper layers, like the administration, might need to know about all remote lus in the system. This interface should be more elaborate and enables to get a list of remote lus according to specific criteria. The criteria can be device type (for example, get a list of all disks in the system), active devices, failed devices and so on.

5.1.1.2. Data members

5.1.1.2.32. Container for all remote lu.

Container for all remote lus in the system.

5.1.1.2.33. Temporary parameters used during lu discovery.

We might need a temporary lu during the discovery process in order to query the device and make sure that this device is not an existing one.

5.1.2. Generic remote lu

Remote lu is an object representing SCSI lu that was discovered over one of the targets presented by the targets manager. The basic type is a generic lu that implements the functionality required by

every SCSI lu as defined by ANSI X3.301 SCSI-3 primary commands (SPC) specification (http://www.t10.org/scsi-3.htm). The generic remote lu is a virtual class without any objects. Each device type (disk, tape) has a different class that is inherited from the generic lu as described in Figure 4.

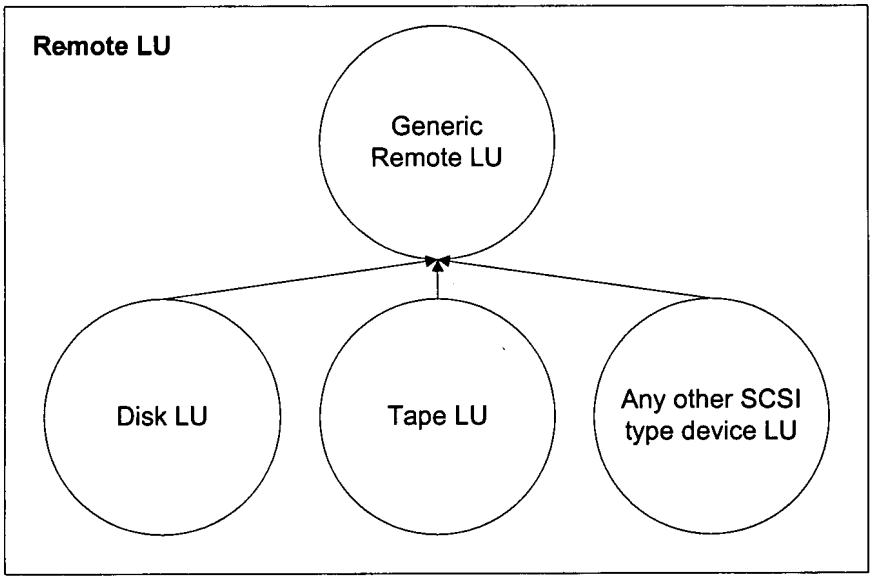

Figure 4 Local LU hierarchy

A derived class from the generic remote lu may use the methods implemented for the basic functionality or override it with device type specific implementations.
Implementation note:
The easiest way to implement a SCSI remote lu is to create a thread of execution for each lu. This thread will handle all requests for the lu. The thread will impose order and priority over the commands and then dispatches them to the physical device through the target object. Error recovery procedures will be performed in an easy way on this thread while blocking all current pending requests. The thread can also periodically check the device and perform proactive operations. In cases of long inactivity periods, the task should initiate power save operations (stop the disk motor for example).

5.1.2.1. Interface

5.1.2.1.34. Get remote lu id

Returns a unique identification of the lu. Unique identification of the lu is a delicate issue and there are several options:
One option is to use the lu number + the target node name in which it resides. This identifies the lu uniquely unless the FC interface of the device is not changed.
Another option is to use the vendor specific identification as returned in the Inquiry data. This option is risky since we depend on non-standard identification of the specific vendor (there are RAID controllers which returns the same identification for all lus in the same controller).
The best option is to rely on standard identification as defined in SPC-2 specification. SPC-2 defines Inquiry page 0x83 as device identification page. The device identification page provides the means to retrieve zero or more identification descriptors applying to the logical unit. Logical units may have more than one identification descriptor. The drawback in this option is that it is still in standardization process and I am not sure whether current available devices support it.

5.1.2.1.35. Get / Set symbolic name

Set or get symbolic name to the lu.

5.1.2.1.36. Get lu capabilities

Returns the lu capabilities that should be exported. At this stage, it looks like most of the device capabilities are not relevant to the user of the lu. Most of the device capabilities are relevant to the operations done inside the lu. Examples for such parameters are queue length and maximum transfer length.

5.1.2.2. Data members

T.B.D

5.1.3. Disk lu

Disk lu is an inherited type of generic remote lu. Disk lu represents any SCSI device that returns Peripheral device type equals to "Direct-Access Device" in it's inquiry data response. This device functionality is defined by T10 SCSI-3 block commands specification (http://www.t10.org/scsi-3.htm). I assume that "disk lu" should be implemented without regards to the disk's vendor. If in a later stage we discover that we have to differentiate between SCSI disk implementations, a derived type will be implemented for each disk vendor (not very likely).
It is worth considering implementing elevator-scheduling algorithm according to the disk geometry. Some experts (Birk) claim that this may enhance disk performance.
The disk object should also keep track of bad sectors on the disk and optionally re-assign them to different sectors if possible.

5.1.3.1. Interface

5.1.3.1.37. Disk constructor

Create a remote disk lu. This operation should include starting the disk, inquiry it's parameters and test it.

5.1.3.1.38. Read / Write sectors

This interface allows sending and receiving data to /from the disk. The parameters to this method are:

- Starting sector.
- Number of sectors.
- SGL pointer for the data.

5.1.3.1.39. Get disk geometry

Returns disk geometry. The parameters that are most relevant to the user of the disk are disk capacity and sector size.

5.1.3.2. Data members

T.B.D.

5.1.4. Tape lu.

Not implemented now.

5.1.5. Remote transaction.

Remote transaction object represents a single command sent form the SDC to one of its attached devices. Remote transactions are generated by one of the remote lus. The remote lu object receives "human understandable" command and translates it to SCSI commands in the form of "remote transactions". The remote transaction is sent from the "remote lu" to the "target" and then to the "target path".
For convenience, the remote transaction object will also include sending task management command options.

5.1.5.1. Interface

5.1.5.1.40. Remote transaction constructor

Create a new transaction or task management.

5.1.5.1.41. Get /set parameters

Get or set one of the transaction parameters (see below).

5.1.5.1.42. Abort transaction

Abort the execution of this transaction.

5.1.5.2. Data members

5.1.5.2.43. Target

5.1.5.2.44. Destination lu number

5.1.5.2.45. Command Descriptor Block (CDB)

5.1.5.2.46. Data length

5.1.5.2.47. Command attribute: head of queue, simple, ordered, aca

5.1.5.2.48. Sense information buffer

5.1.5.2.49. Timeout value for the command

5.1.5.2.50. Pointer to scatter gather list with the data of the command

5.1.5.2.51. Data direction (read, write, none)

5.1.5.2.52. Command priority

5.1.5.2.53. Task management

6. SCSI Router

The SCSI router performs a "one-to-one" bridging between FC/SCSI devices to their representation on the iSCSI interface. According to the configuration, selected targets seen by the FC/SCSI interfaces are presented to the iSCSI as the same targets. Commands arriving on the iSCSI interface to these targets are sent directly to the device on the SCSI/FC interfaces. The SCSI router has no notion regarding logical units within the targets. The router routes SCSI information between SCSI-3 targets seen on the Fibre channel interfaces to SCSI targets on the iSCSI interface. The router doesn't implement a "SCSI-3 device server" thus does not have to deal with commands ordering and dispatching.

The iSCSI module is responsible to stripe off any information that is not pure SCSI-3 before sending it to the router on one hand and encapsulate pure SCSI information coming from the driver with iSCSI headers on the other.

The fibre channel driver is responsible to encapsulate pure SCSI information with fibre channel headers on one hand and stripe it off before sending it to the router on the other.

Each command starts with an iSCSI command that encapsulates SCSI CDB and SCSI command parameters. The CDB and command parameters are taken from the iSCSI header and sent to the router. The router finds the appropriate device on the fibre channel side and sends the same CDB and parameters to the FC driver. The FC driver creates an FC command and embeds in it these parameters. If this command is a write command, the next step is to receive an RTT from the FC driver and to send it through the router to the iSCSI module. It is important to mention that there are no RTTs in parallel SCSI thus the router should be more sophisticated and fake RTTs to the iSCSI in order to get the data. The next stage is to transfer data to or from the device through the router. The last stage in a command is the SCSI response. SCSI response from the FC device is sent to the iSCSI module were it is encapsulated with iSCSI header and sent to the command originator.

The router should be as transparent as possible and perform the minimal translations needed. These translation operations mainly include address conversion and data segmentation. There are also some advanced issues which should be addressed like third party operation which include addresses of a third party device embedded in the command. Theses third party addresses should be converted from iSCSI notation to fibre channel addresses. Figure 5 describes the interface between the SCSI router and the iSCSI and FC modules.

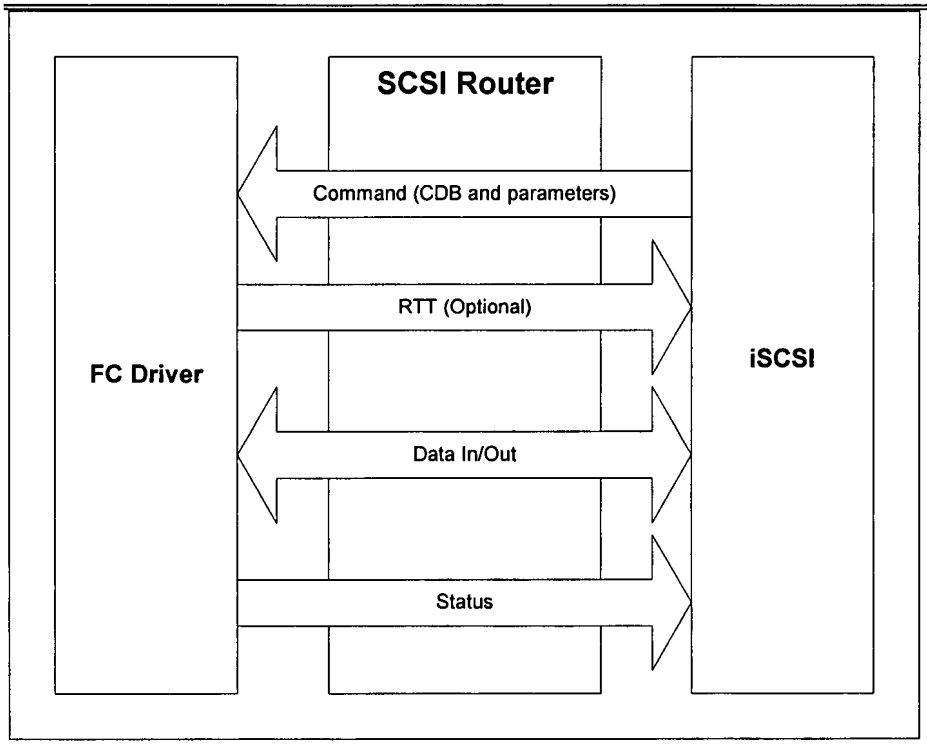

Figure 5 SCSI Router

Remark: At this stage SANRAD will not implement a SCSI router.

7. Local LU Manager

The local lu manager module implements virtual lus (disks, tapes and any other SCSI device). The different lus are exposed to selected initiators (according to the configuration) on the iSCSI interface. The module should interface to any SCSI target mode implementation but in our case, the target mode is implemented over iSCSI transport. The module should adhere to SCSI-3 Architecture Model (SAM) specification as defined by T10 committee (http://www.t10.org/scsi-3.htm).
At this stage, only "disk lus" will be supported. Each volume created by the user will be exposed on the iSCSI network as a different lu with unique lu number. This means that at this stage, the SDC will be identified as one target with multiple lus. According to the configuration, the lu will be presented only to those initiators that were granted the permission to use it.

At the end of this section, there is an example of local lus manager objects and their relationship (see Figure 7 Local Lus Manager Example).

7.1. Basic objects

7.1.1. Local lu manager

The local lu manager object is responsible for creating new lus delete existing lus, dispatching new commands to the appropriate lu and handle task management commands. The local lu manager is a single object in the system.

7.1.1.1. Interface

7.1.1.1.54. Local lu manager constructor

Create the local lu manager and init it's data members.

7.1.1.1.55. Create lu

Creating new lu is issued from the administration module. In most cases, the administration module gets requests from the management and among other operations, creates the new lu. There are different methods for different lu types. For example, creating a new volume requires additional parameter for the volume handle while creating a new tape might require different parameters. There are common parameters for all lu types like:

- *Lu number*: 8 bytes indicating the lu number.
- *Queue length*: restriction for the number of commands that can be executed concurrently.
- *Access permission list*: list of all initiators that have access permission for this lu including differentiation between read only and read/write permission. The access permission might be implemented as a function pointer to a "user supplied" access permission function. In this case, the module that created the lu has more refined control over the access permission.

7.1.1.1.56. Delete lu

Deleting lu is not a common operation and should require more caution. Before we start the operation we should guarantee that there are no commands pending and block all new commands. It is worth to consider generating a soft reset after such an operation.

7.1.1.1.57. Receive command

The iSCSI module has no notion regarding lus within the SDC. It is the responsibility of the local lu manager to match between the lu number and the actual local lu and send it to it. It is up to the local lus designer to decide whether to allocate the command resources in the local lu manger or in the lu itself. In a perfect world, the allocation should be per lu thus having more control over the lu queue.

7.1.1.1.58. Receive task management command

ISCSI module sends task management commands to the local lu manager. The local lu manager dispatches it between the lus and waits that all commands affected by this task will be terminated. The following task management functions should be supported:

- Abort task: abort a specific task.
- Abort task set: abort all tasks in the task set for the requesting initiator.
- Clear task set: abort all tasks in the task set for all initiators.
- Target reset: reset the target device and terminate all tasks in the task sets of all lus.

The lu manager notifies the different lus which tasks should be aborted and waits for a response. When all tasks are killed, a task management response is sent back to the iSCSI module and an iSCSI tasks management response frame is sent back to the requesting initiator. We should further investigate which cases require sending asynchronous events to all other initiators that use this lu.

7.1.1.2. Data members

7.1.1.2.59. Lus container

Container of all lus in the system.

7.1.1.2.60. Lus dispatch table

Table that maps lu numbers to local lu objects.

7.1.2. Generic local lu

Local lu is an object representing SCSI lu that is exposed over the iSCSI interface to selected iSCSI initiators. The basic type is a generic lu that implements the functionality required by every SCSI lu as defined by ANSI X3.301 SCSI-3 primary commands (SPC) specification (http://www.t10.org/scsi-3.htm). The generic local lu is a virtual type without any objects. Each device type (disk, tape) has a different class that is inherited from the generic lu as described in Figure 6 Local LU hierarchy.

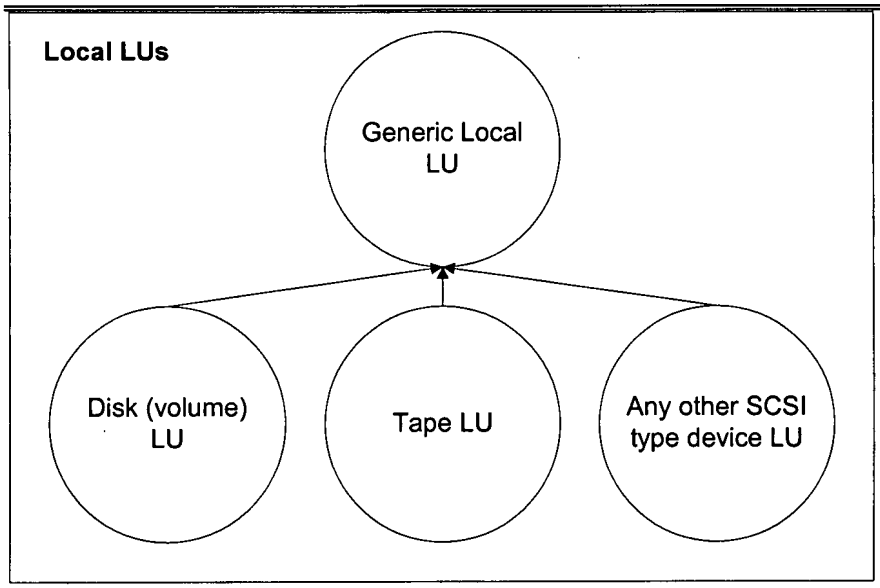

Figure 6 Local LU hierarchy

7.1.2.1. Interface

7.1.2.1.61. Receive new command

New commands are dispatched from the lu manager to the lu. All SCSI device types must support the following commands:

- Inquiry.
- Request sense.
- Mode sense.
- Mode select.
- Test unit ready.
- Report luns.

Different device types support additional commands according to the T10 specifications. Device classes might also overload the implementation of the basic commands in the generic lu class.

Access control to the lu can be implemented in the generic lu class. For each command arriving to the lu, we have to check if the initiator that sent it has the appropriate access permission and if not, reject the command with access violation sense info. The type of the command (read/write) can be identified using special bits without parsing the CDB.

7.1.2.1.62. Change access permissions

Enables the configuration to change the access permission list for this lu even "on the fly".

7.1.2.2. Data members

7.1.2.2.63. Lu number.

Eight bytes dentifying the local lu number.

7.1.2.2.64. Exec local transaction queue

Queue of commands that are currently executing and waiting for completion.

7.1.2.2.65. Dormant local transaction queue

Queue of commands that were dispatched from the local lu manager but still can't be executed (simple and ordered commands).

7.1.2.2.66. Access control list

List of all initiator and their access control to this lu.

7.1.3. Disk lu

Disk lus should adhere to SCSI-3 block commands (SBC) T10 specification (http://www.t10.org/scsi-3.htm). The most important commands that we should impellent are:

- Read6, Read10.
- Write6, Write10.
- Read capacity.

7.1.3.1. Data members

- Capacity
- Block size
- Handle to the volume.
- Virtual disk capabilities used to implement Inquiry and mode sense.
- Locked areas map (optional).

7.1.4. Tape lu

Not supported at this stage.

7.1.5. Local transaction

Local transaction object represent a single SCSI command that arrived on the iSCSI interface. This object "lives" during the time the command is executed and freed only when it is finished.
Each local transaction object is bounded to one thread of execution in which the command is executed.
Each local transaction object embeds a data transfer object that is responsible for the actual data movement between the iSCSI module and the attached devices.

The "local transaction" object holds all the necessary information regarding the command like:

- The lu number to which the command was sent.
- Command Descriptor Block (CDB).
- Expected data length.
- Command attribute: head of queue, simple, ordered, aca.
- Handle to the transaction at iSCSI module.
- Handle to the initiator that sent the command.

The object also includes various data members that may help during the execution of the command and will save allocating resources on the fly like:

- Scratch data for building sense information.
- Pointers to hold data scatter gather lists.

- Temporary semaphores and message queues.
- Data transfer object.

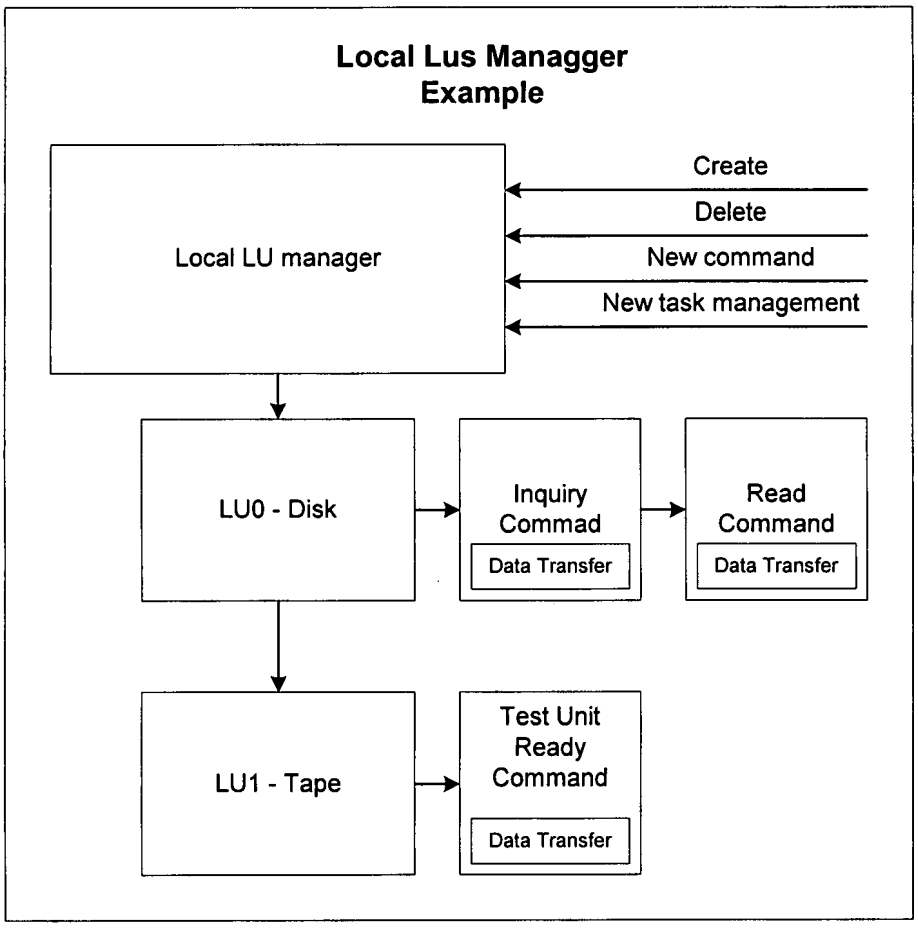

Figure 7 Local Lus Manager Example

8. Volume virtualization

This module is responsible for volume virtualization operations. The module is at the heart of the virtualization system and all other modules are providing services to the module and connecting it to the rest of the system. To clarify what virtualization means, we can say that it is a mapping scheme (most of the time it is static) between virtual disks to physical disks. The virtual disks are created and managed by the local lu manager and the physical disks are discovered and managed by the remote lu manager. Each request to the virtual volume is directed to this module and this module provides the actual commands that should be sent to the physical disks. There are two options to implement this module. The first one is to send a request to this module and let it handle the whole command including performing the actual data transfer. This implementation is more standard and is used by most volume managers. The second option is to use this module as a consulting module. In this implementation, we send a command to the module and get in return a list of commands that should be performed on the physical disks. We have decided to implement the module as a consulting module for the following reasons:

- The virtualization module should not have any knowledge regarding "how to exchange the data" it should only provide an answer to the question "where to put / get the data".
- It is more efficient to tell the iSCSI module the mapping of the whole command instead of telling it where to put the data for each "segment" of the command.
- Implementing the virtualization as a consulting, separate module will enable you to import this module easily to other virtualization products (box X).

8.1. Basic objects

8.1.1. Volume

Volume is a basic virtual type for all volume types implemented in the system.
Volume is defined as a continuous data blocks with the same block size. The user of the volume can address all blocks up to the volume capacity. The volume might be spread over different physical disks according to the virtualization techniques used. Figure 8 describes the different volume types according to the virtualization technique used:

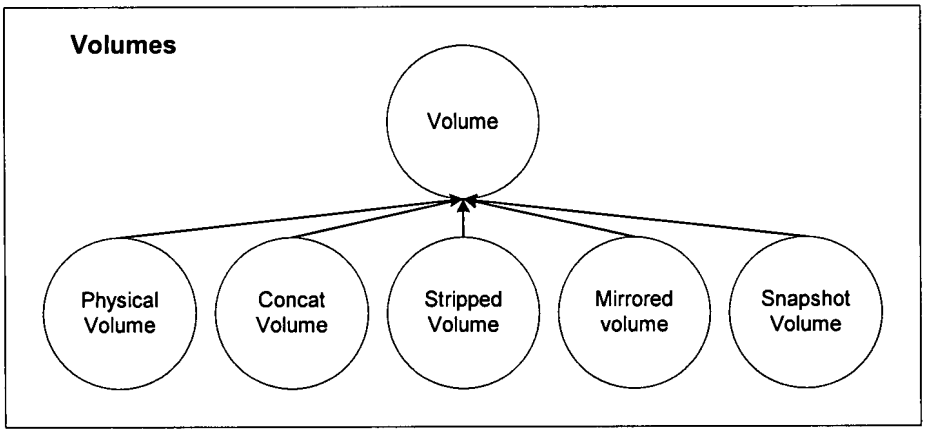

Figure 8 Volumes hierarchy

A volume can be created from physical volumes (disks RAIDs) or from already created volumes. This way, we can create a volume that employs more than one virtualization technique. For

example, a volume can be a mirror set of two other volumes, one is a stripe over 3 disks and the other is a concatenation of two disks as described in Figure 9.

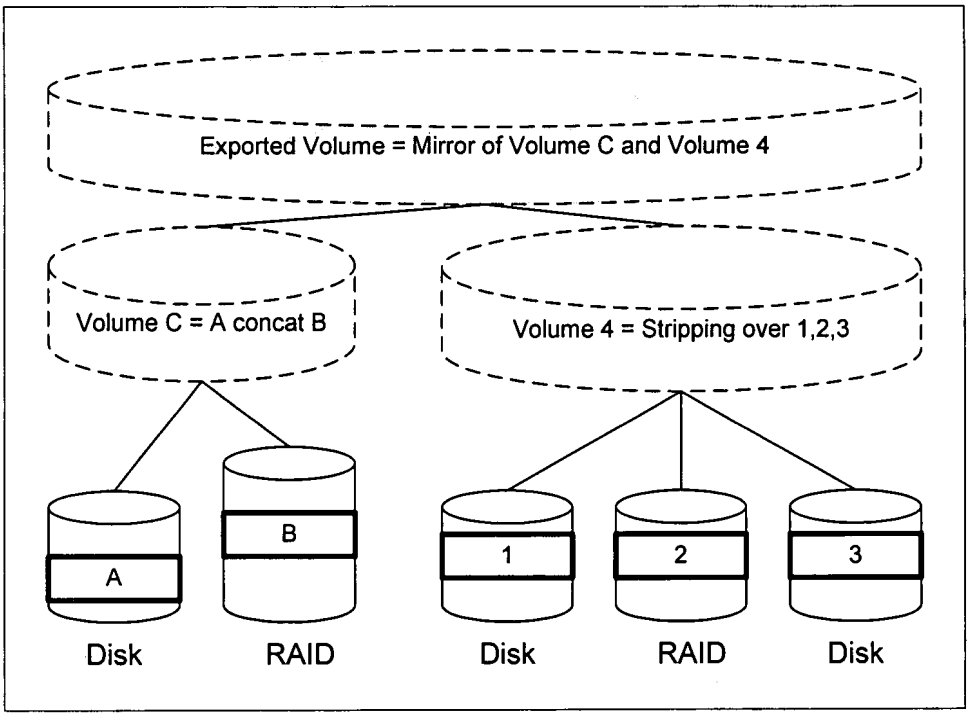

Figure 9 Volume example

8.1.1.1. Interface

8.1.1.1.67. Volume constructor

Create a new volume using a list of either disk segments or already created volumes.

8.1.1.1.68. Volume destructor

Delete an already created volume.

8.1.1.1.69. Consult read

Get a list of commands satisfying this read request.

8.1.1.1.70. Consult write

Get a list of commands satisfying this write request.

8.1.1.1.71. Get capacity

Returns volume capacity in blocks.

8.1.1.1.72. Get block size

Returns volume block size in bytes.

8.1.1.2. Data members

T.B.D

The following gives a brief description of each volume type:

8.1.2. Physical volume

Physical volume is a continuous area of sectors over one of the remote lu disks. This is the building block of all other volume types.
The disk handle of the disk is provided to this object when it is created. This handle will further identify the disk when the data transfer engine starts actual data transfer.

8.1.3. Concatenation volume

A volume consists of two or more volumes that can be of different sizes, concatenated together to create a single volume. The first blocks of the volume reside in the first volume; the following blocks are in the next volume and so on.

8.1.4. Stripped volume.

Created from two or more volumes of the same size. The data is stripped over the volumes according to the "stripe size".

8.1.5. Mirrored volume.

Created from two or more volumes of the same size. Each volume contains a complete copy of the data. Writes are directed to all volumes while reads are directed only to one of the copies.

8.1.6. Snapshot volume.

T.B.D (see snapshot design).

8.1.7. Volume response

Volume response is the return value of invoking "consult read" or "consult write" on a volume. The response is a list of read/write commands to be taken over physical disks. As described earlier, the disks are the building blocks of the physical volumes that are leafs in the volume tree hierarchy. The commands are not SCSI commands but contains the following information:

- Read or write.
- Source data offset in the received data.
- Data length.
- Disk handle.
- Starting sector in the disk (start LBA).
- Number of sectors.

The list of commands should impose some ordering. There are commands that can be executed concurrently while others must be performed sequentially.

8.1.7.1. Interface

T.B.D.

8.1.7.2. Data members

T.B.D.

9. Data Transfer Engine

See Erez's design.

10. Configuration

Remark: This module is a "system module" and is not in the scope of virtualization.

The configuration module saves, restores and maintains permanent configuration parameters from all modules in the system.
This module provides a generic interface for any module or application in the system to save permanent parameters that are consistent between SDCs boot-ups.

IMPORTANT: Configuration parameters are common for all Storage Domain Controllers (SDCs) in the domain. Changing a parameter should first be negotiated between all controllers in the domain and than be saved in a common configuration database.

I don't intend to describe a full mechanism but only to provide a few ideas:

1. The permanent data can be saved on disks attached to the SDCs domain.
2. We should save some space in all disks for configuration information.
3. We must have more than one copy of the configuration.
4. We would like the configuration to be secured and encrypted.
5. We would like to have an option to save the configuration on an external media.
6. We can't write to a disk less than sector size bytes so one simple solution is to have a copy of all parameters in the SDC memory and always save the whole data.
7. All SDCs in a domain should be notified regarding configuration change.
8. We can save the configuration parameters in text format (like old .ini files).

10.1. Virtualization parameters

10.1.1. Remote lus

10.1.1.1. Disks

- Disk identification.
- Bad sectors list.

10.1.2. Local lus

10.1.2.1. Volumes

- Volume structure.
- Volume access permissions list.

11. Intra Domain Communication

T.B.D.

12. Administration

Remark: This module is a "system module" and is partially in the scope of virtualization.

The administration module performs, monitors and synchronizes the actual execution of each configuration/operational changes in the system.
The administration module does not deal with normal data flow but only handles events of configuration change and error condition.
We can say that the administration acts as a linking link between the configuration/management and the reset of the system. Configuration changes are made in the configuration module and submitted to the administration for execution. On the other hand, normal or abnormal events in the system are reported to the administration and among other things, the administration reports it to the management (to the user).
For each such event, a predefined sequence of operations is performed. It is recommended that the module will handle only one event at a time with some priority between events.

The following are the main "virtualization events":

12.1. Creation of a new volume

12.2. Creation of a volume after reboot

12.3. Deletion of a volume

12.4. Change volume parameters

12.5. Volume failure

13. Initialization

Remark: This module is a "system module" and is partially in the scope of virtualization.

The initialization module boots the system and creates all required objects according to the configuration information. Since the configuration data resides in the disks attached to the SDC, the first module to be loaded is the device manager. After the device manager is loaded, we can create the configuration module. The configuration module checks in all disks for a valid configuration file and read it to the memory. Now the administration, local lu manager, iSCSI and management are booted. The initialization module reads the configuration and starts building all volumes through the administration.

Remark: In a single domain multiple SDCs configuration we might want to first boot the administration and intra domain communication modules before we read the configuration.

Remark: You should also think about how to "register" this new booted SDC in the domain. All other SDCs in the domain should know that a new SDC is powered on and we must make sure that all SDCs "see" the same configuration information. Another question to be asked is how do we define "all other SDCs in the domain?"

14. Management

Remark: This module is a "system module" and is not in the scope of virtualization.

T.B.D.

15. System utilities

Remark: This module is a "system module" and is not in the scope of virtualization.

15.1. Memory manager

15.2. Linked lists

15.3. Scatters gather lists (SGL)

15.4. Memory pools

Appendices

Index

Change History

Date of issue	Revision	Author	Requirements Manager	Change Summary	Page Count
9/6/2007	1.0	Sarel Altshuler	Sarel Altshuler		39